Improving Multi agent Systems Based on Reinforcement Learning and Case Base Reasoning

Sara Esfandiari[1], Behrooz Masoumi[1], Mohammad Reza Meybodi[2], Abdolkarim Niazi[3]

[1] Department of Computer Engineering and Information Technology, Islamic Azad University, Qazvin Branch, Qazvin, Iran

[2] Departments of Computer Engineering, Amirkabir Industrial University, Tehran, Iran,

[3] Department of Manufacturing and Industrial Engineering, Faculty of Mechanical Engineering, Universiti Teknologi Malaysia, 81310 UTM Skudai, Malaysia

Abstract

In this paper, a new algorithm based on case base reasoning and reinforcement learning is proposed to increase the rate convergence of the Selfish Q-Learning algorithms in multi-agent systems. In the propose method, we investigate how making improved action selection in reinforcement learning (RL) algorithm. In the proposed method, the new combined model using case base reasoning systems and a new optimized function has been proposed to select the action, which has led to an increase in algorithms based on Selfish Q-learning. The algorithm mentioned has been used for solving the problem of cooperative Markov's games as one of the models of Markov based multi-agent systems. The results of experiments on two ground have shown that the proposed algorithm perform better than the existing algorithms in terms of speed and accuracy of reaching the optimal policy**.**

Keywords: *Reinforcement learning, Selfish Q-learning, Case-base reasoning Systems, Multi-agent Systems, Cooperative Markov Games.*

1. Introduction

Case Based Reasoning (CBR) is a knowledge based problem solving technique, which is based on reusing on the previous experiences and has been originated from the researches of cognitive sciences [1]. In this method, it is assumed that the similar problems can possess similar solutions. Therefore, the new problems may be solvable using the experienced solutions to the previous similar problems. A multi-agent system (MAS) is comprised of a collection of intelligent agents that interact with each other in an environment to optimize a performance measure [2]. Agents are computational entities that can see their environments with their sensors. These agents should do appropriate action in per moment based on their observations. In multi agent system research, cooperative and non-cooperative perspective. In cooperative multi-agent systems, the agents pursue a common goal and the agents can be built expect benevolent intentions from other agents. In contrast, a non-cooperative multi agent system setting has non-aligned goals, and individual agents try to obtain only to maximize their own profits. In multi-agent systems, the need for learning and adoption is essentially caused by the fact that the environment of the agent is dynamic and just empirically observed while the environment (the reward functions and the transition states) is unknown. Hence, the reinforcement learning methods may be applied in MAS to find an optimal policy in MGs. In addition, agents in a multi-agent system face the problem of incomplete information with respect to the action choice. If agents get information about their own choice of action as well as that of the others, then we have joint action learning [3][4]. Joint action learners are able to maintain models of the strategy of others, and the explicitly takes into account the effects of joint actions. In contrast, independent agents only know their own action which is often a more realistic assumption since distributed multi-agent applications are typically subject to limitations such as partial observability, communication costs, and stochastic.

There are several models proposed in the literatures for multi-agent systems based on Markov models. One of these models is stochastic games (also called Markov Game – MG). Markov games are extensions of Markov

Decision Process (MDP) to multiple agents. In an MG, actions are the result of joint action selection of all agents, while rewards and the state transitions depend on these joint actions. In a fully cooperative MG called a multi-agent MDP (or MMDP), all agents share the same reward function and they should learn to agree on the same optimal policy [5].
There are several methods for finding an optimal policy in MMDPs. In [6], an algorithm is proposed for learning cooperative MMDPs, but it is only suitable for deterministic environments. In [7] an algorithm called *Selfish Q-Learning* has been introduced which changes the Q values of each action used a special Q-function. In [8] MMDPs are approximated as a sequence of intermediate games. The authors present optimal adaptive learning and prove convergence to Nash equilibrium of the game. In [9], an algorithm called *CAQL* has been introduced, which acts through *a Q - learning algorithm*. In [10], a Q-learning algorithm based method has been proposed.
In Reinforcement Learning (RL), learning is carried out online, through trial-and-error interactions of the agent with the environment. Unfortunately, convergence of any RL algorithm may only be achieved after extensive exploration of the state-action space, which can be very time consuming. However, the rate of convergence of an RL algorithm can be increased by using heuristic functions for selecting actions in order to guide the exploration of the state-action space in a useful way. In [11], [12] investigates how to make improved action selection functions based on heuristics in on-line policy learning for robotic scenarios. These functions have been applied to select the action in every state. Although these methods have been successfully used to find the optimal policy in Markov games, the problem of using the previous experiences of agents for solving the new problem is still disregarded in these methods. Since in the environment is unknown in multi-agent systems, and the agent should upgrade its knowledge of environment through observation, so the problem of keeping and reusing the previously acquired knowledge causes an increase in learning rate. In this paper, to increase the speed of learning rate to get the optimal policy for Markov Games in the independent agent's state, a hybrid algorithm called Case-based Best Heuristically Accelerated Selfish Q-learning (CB-BHASQL) is proposed in which, a modified function is used to select the action and the Case Base Reasoning technique and a special Q-function called Selfish Q-Learning has been used to increase the learning rate. To evaluate the proposed methods, they have been applied to two examples of MMDP called Grid Game and Tunnel To Goal. The results of computer simulations have shown that these algorithms outperform the previous approaches from both cost and speed perspective. In the next part of the paper, at first fundamental concepts are explained in section 2 and in section 3, the proposed algorithm is presented. Simulation results, and discussions are reported in section 4 and in section 5, evaluation of the algorithm's behavior and its analysis is done and section 6 is the conclusion.

2. Reinforcement Learning

In this section, we first review some basic principles of Markov decision Process (MDP) and then present the basic formulation of the Q-learning algorithm, a well-known reinforcement learning technique for solving MDPs. A reinforcement learning agent defines its behavior through interaction with an unknown environment and observation of the results of its behavior [12].

2.1 Markov decision Process

Markov decision process is formally defined as follows:

Definition 1. *A Markov decision process (MDP) is a quadruple ⟨S, A, R, T (where S is a finite state space; A is the space of actions the agent can take; R: S×A (ℜ is a payoff function (R (s, a) is the expected payoff for taking action an in state s); and T: S×A×S ([0,1] is a transition function (T (s, a, s') is the probability of ending in state s', given that action a is taken in state s).*

In a Markov decision process, an agent's objective is to find a strategy (policy) π*: S →A* so as to maximize the sum of discounted expected rewards,

$$V(s, \pi) = \sum_{t=0}^{\infty} \gamma^t E(r_t | \pi, s_0 = s) \quad (1)$$

Where s is a particular state, s_0 indicates the initial state, r_t is the reward at time t, and $\gamma \in [0,1)$ is the discount factor. There exists an optimal policy π^* such that for any state s, the following equation holds:

$$V(s, \pi^*) = max_a\{r(s,a) + \gamma \sum_{s'} P(s'|s,)v(s',\pi^*)\} \quad (2)$$

where $r(s, a)$ is the reward for taking action a at state s, and $v(s,\pi^*)$ is called *optimal value* for that state while $P(s_0|s, a)$ is the probability of transiting to state s' After taking action *an* in state s. If the agent knows the reward and state transition functions, it can solve π^* by iterative search method, otherwise this method cannot be used while an algorithm called Q is employed [13] [14].
The variety of Q-functions has been used in [6,7,8,9,10]. Selfish Q-learning algorithm pseudo-code, which has been used in [7], is shown in Figure 1. In this algorithm, for every action a in each state S the value of that action ($Q(s, a)$) is used according to Equation 3. Each state S the value of that action ($Q(s, a)$) is used according to

IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 3, January 2012
ISSN (Online): 1694-0814
www.IJCSI.org

Equation 3. In Equation3, α is the rate of learning and $\gamma \in [0, 1]$ is the discount factor. The algorithm ends when the optimum policy doesn't change for a definite while.

$$Q(S_t, a_t) = Q(S_t, a_t) + \alpha[\, r_t + (\gamma \max Q(S', a')) - Q(S_t, a_t)] \quad (3)$$

To select an action in every state, the Boltzmann's distribution method (EQ 4) is usually used. This function has been used in some articles such as [6,7,8,9,10,19].

$$\pi_1(S) = \arg\max \left(\frac{e^{\frac{Q(S,t)}{\tau}}}{\sum_{i=1}^{m} e^{\frac{Q(S,t)}{\tau}}}\right) \quad (4)$$

In which, m is the number of allowable actions for state S and τ is a constant. Q (S, a) shows the value of evaluation function of state S while action a is done.

Algorithm Selfish Q-Learning
1. **Initialize** Q(S,a) arbitrarily
2. **Repeat** (**for each** *episode*)
3. **Initialize** S randomly
4. **Repeat** (**for each** step)
***5.* Select** an action using $\pi_1(S) = \arg\max \left(\frac{e^{\frac{Q(S,t)}{\tau}}}{\sum_{i=1}^{m} e^{\frac{Q(S,t)}{\tau}}}\right)$ **EQ(4)**
6. **Execute** the action a
7. **Observe** reward *r(s,a) ,state s'*
8. **Update** the value of *Q(S,a)* according to $Q(S_t, a_t) = Q(S_t, a_t) + \alpha[\, r_t + (\gamma \max Q(S', a')) - Q(S_t, a_t)]$ **EQ(3)**
9. $S \leftarrow S'$
10. **Until** S is *Terminal State*
11. **Until** some stopping Criteria is reached.
12. End

Fig 1. Selfish Q-Learning Algorithm

2.2. Markov Games

Markov games are a generalization of MDPs to multiple agents and can be used as a framework for investigating multi-agent learning. In the general case (general-sum games), each player would have a separate payoffs. A standard formal definition follows:

Definition 2. *A stochastic game (Markov game) is a tuple* $\langle n, S, A_{1..n}, T, R_{1..n} \rangle$*, where n is the number of agents, s is a set of states,* A_i *is the set of actions available to agent i (and A is the joint action space* $A_1 \times A_2 \times \ldots \times A_n$ *), T is a transition function* $S \times A \times S \rightarrow [0,1]$ *), and r is a reward function for the* i_{th} *agent* $S \times A \rightarrow \Re$.

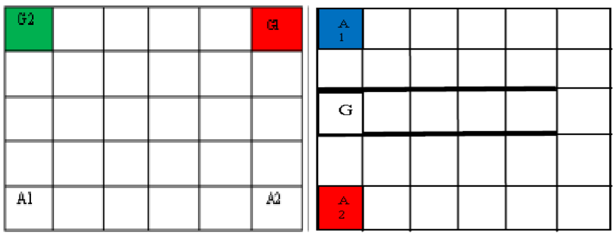
In a discounted Markov game, the objective of each player is to maximize the discounted sum of rewards, with a discount factor $\gamma \in [0,1)$. Let π_i be the strategy of the player i. For a given initial state s, player i tries to maximize:

$$v(s, \pi^1, \pi^2, \ldots, \pi^n) = \sum_{t=0}^{\infty} \gamma^t E(r_t | \pi^1, \pi^2, \ldots, \pi^n, s_0 = s) \quad (5)$$

Markov games are categorized based on the agent's rewards into cooperative and non-cooperative games. Non-cooperative games may be classified as competitive games and general-sum games. Strictly competitive games, or zero-sum games, are two-player games where one player's reward is always the negative of the others. General-sum games are ones where the reward sum is not restricted to zero or any constant, and allow the agents' rewards to be arbitrarily related. However, in full cooperative games, or team games, rewards are always positively related. In a fully cooperative MG (or team MG) called a multi-agent MDP (or MMDP), all agents share the same reward function. Nevertheless, in general MG (or general-sum MG) there is no constraint on the sum of the agents' rewards and the agents should learn to find and agree on the same optimal policy. However, in a general Markov Game, an equilibrium point is sought; i.e. a situation in which no agent alone can change its policy to improve its reward when all other agents keep their policy fixed [15], [16].
One of the Markov's games used for multi-agent Markov's games is the Grid World game. In this game, two agents start from a corner of the page and try to reach a goal with the least possible number of moves. Players' actions are defined as four actions in four different directions, namely Up, Down, Left, Right. A state space set is defined as $S = \{s | s = (l_1, l_2)\}$, In which each state s$=(l_1, l_2)$ Indicates the coordinates of agents 1 and 2. Agents cannot take the same coordinates at the same time. In other words, if both agents try to move to the same square, both of their moves will fail. If agents move to two different non-goal positions, both receive zero rewards and if one reaches the goal position, it receives 100 units of reward. However, if they collide with each other both receive one unit of punishment and stay in their previous position. In this game, the state transition is deterministic, i.e. the next state is uniquely determined by the current state and the joint action of the agents. In this game, agents are assumed not to know the goal position and the other agent's reward functions. Agents choose their actions simultaneously and can only know about the previous moves of the other agents and their own current state.

Another game which we have used it, called *Tunnel to Goal*. In this game, there are some barriers. If an agent collision these barriers, it receives one unit of punishment.

A path in these games represents sequences of actions from the starting to the end position. In game terminology, such a path is called a policy or strategy. The shortest path, not interfering the path taken by the other agent, is called the optimal policy or *Nash path*. Figure 2 is an example of these games. The optimal policy in Figure 2.a includes 9 movements and in Figure 2.b includes 10 movements.



(a) (b)

Fig2 . examples of Markov Games. (2.a) An example of Grid World Game. 2.b. An example of Tunnel to Goal Game

2.3. Case Base Reasoning

Case Based Reasoning (CBR) technique uses the previous experiences (Case) to solve the new problems [17], [18]. In the case base reasoning systems, the experiences gained from solving the problems are saved in case base (CB). In these systems, for solving the new problem (C_{new}), the most similar cases to C_{new} are extracted from the case base (CB) and the solutions presented by the extracted cases are used to solve the new problem C_{new}. If a similar case is not found, C_{new} is inserted is inserted to the case base as a new case. Unlike the classical knowledge-based methods, CBR focuses on a particular problem-solving experience, which is originated from the cases collected in the case base. These cases show a particular experience on a problem solving domain. It must be noted that CBR doesn't recommend a definite solution, but presents hypothesis and theories pass the solution space.

3. The Proposed method

In this section, a new algorithm called *CB-BHASQL* is proposed to increase the rate of convergence in Markov's games. In the proposed algorithm, the case base reasoning and also a new function are used to select the action in each state to increase the convergence rate toward the optimal policy. We previously used our new function with Decentralized Q-Learning according to EQ(6) and called CB-BHADQL. In this paper, We proposed a new algorithm with Selfish Q-Learning according to EQ(3) and called CB-BHASQL.

$$Q(S,a) = (1-\alpha)Q(S,a) + \alpha(r + \gamma \max_b Q'(S,a)) \quad (6)$$

We know that solving a problem using CBR includes the steps: creating a description of the problem, evaluating the similarity of the current problem to the previously-solved problems saved in case base, and trying to reuse the solutions presented by the detected cases to solve the current problem. The structure of the cases used in the recommended algorithm is a duplex in the form of *Case=<Prob, Sol>* in which, *Prob* describes the problem and *Sol* is the solution presented to solve the problem. The problem describer (*Prob*) includes the properties in each state. In the proposed algorithm, the problem describer is defined as *Prob (S) = {m, <Up, Down, Right, Left>, index}* in which, *m* is the number of actions for each state and the set *<Up, Down, Right, Left>* are the actions allowable for each action and *index* is the index for each state. The solution recommended for the problem is *Sol (S) = <E, V>*, in which vector $\vec{E}$ In the form of $\vec{E}$= ($\vec{E}$*[1]*, $\vec{E}$ *[2], ...,* $\vec{E}$ *[m]*) is a list of experiences collected from the environment by the agent for state *S* and each vector $\vec{E}$ includes a tuple $<A_i, N_i, Q_i, \pi_i>$ where A_i is the space of actions for state *S* and N_i is the number of times that $a_i \in A_i$ has been updated and Q_i is the value estimated by Equation 1 and π_i is the possibility of occurrence of action a_i , which is estimated by EQ(7).

$$\pi_2(S) = \arg\max\left(\frac{e^{n(S,a)Q(S,a)}}{\sum_{i=1}^{m} e^{n(S,a)Q(S,a)}}\right) \quad (7)$$

Where *m* is the number of allowable actions for state *S* and *n* (*S*, *a*) is the number of times that so far the action *a* has been selected. *Q* (*S*, *a*) shows the value of evaluation function of state S while action *a* is done.

V is the justification of using the solution recommended by the detected agent and if each of the actions of state *S* at least has been selected once, the solution of the detected agent can be used for solving the new problem. In the recommended algorithm, once the agent enters a new state, extracts the most similar case to the new state of the case base and if the justification is available *(V = True)*, the detected case is used to determine the next state.

To detect the similar cases of current state, the nearest neighbor algorithm is used. Euclidean distance of the new case to each of the cases available in case base is calculated according to EQ(8) and the most similar case (*c*) is detected and if the justification is available *(V = True)*, the solution of detected case is used to solve the new problem. The proposed algorithm is shown in Figure 3.

$$NN(S) = \arg max_{c \in CB}\, Sim(C.prob, C.sol) = \arg \underset{c \in CB}{max}\, dist(C.prob, C.sol) \qquad (8)$$

4. Experiments

In order to evaluate the performance of the proposed algorithm several experiments have been conducted whose results are reported below. In Section 4.1, The environment of the experiments is a Grid-world game that includes a 5 ×6 Grid according to Figure 2.a. In section 4.2, the environment of the experiments is a Tunnel to Goal game that includes a 5 ×6 Grid according to Figure 2.b. These experiments are conducted to study the improvement obtained by the proposed algorithm (CB-BHASQL) in comparison with three CBR and QL algorithms and CB-BHADQL. So, CB-BHASQL algorithm is compared with three algorithms: 1) Selfish Q-Learning algorithm, and 2) Boltzmann's CBR algorithm, which its pseudo-code is similar to Figure 3 and the only difference is in the selection of the actions which is based on the Boltzmann's distribution (Equation 4) and 3) CB-BHADQL. In all experiments, each reported value is obtained by averaging over 200 runs and the average results are gained for the algorithms. Parameters given are $\tau = 0.05$ and $\gamma = 0.7$.

4.1. Experiments in Grid World Games

In this section, we show results of our experiments in Grid word Games.

Experiment 1. In this experiment, we compare the proposed algorithm (CB-BHASQL) in Grid World Games environment (Fig 2.a) with the other algorithms in terms of the number of movements made by agent 1 to reach the optimal path in 2000 episode. Figure 4 illustrates the results of this experiment. Figure 5 shows the average results after 200 runs. From the result, it is evident that the CB-BHASQL algorithm has lower numbers of moves in comparison with the other algorithms.

Experiment 2. In this experiment, we compare the proposed algorithm (CB-BHASQL) in Grid World Games environment (Fig 2.a) with the other algorithms in terms of the averaged reward received by agent 1during an episode. Figure 6 shows the result of this experiment. As it is seen (CB-BHASQL) algorithm outperforms the other algorithms in terms the average reward received during an episode.

Algorithm CB-BHASQL

Let t be the global time, n be the number of agents, γ the discount factor ,$CB_i=\phi$ an empty case base for each Set $s = s' \in S$to the initial state of the system

Repeat

Set $s=s'$

forall agent $i \in [1 \ldots n]$ do

if $CB_i=\phi$ or **addcasecriterion(s)** *is true*

$CB = CB \cup C$ with *c.Prob=s* and *c.Sol=empty_solution (i)*

for each j=Sol(s).m do

Compute *Sol(s).E[j].* π_i according to **EQ(7)** and **Set** index x_i the *Maximum value of them.*

Select *elementary action* $Sol(s).E[x_i].a_i$.

Observe *Successor state* $s' \in S$ and *reward* $r \in R$.

end if.

end for.

for all agents $i \in [1 \ldots n]$ do

Retrieve *nearest neighbour* according to **EQ(8)** of state *s'*.

Set *Learning Rate* $\alpha_i = \dfrac{1}{1 + Sol(s).E[x_i].n_i}$

Set $Sol(s).E[x_i].Q_i$ according to $Q(S_t, a_t) = Q(S_t, a_t) + \alpha[\, r_t + (\gamma \max Q(S', a')) - Q(S_t, a_t)]$ **EQ(3)** .

Increment $Sol(s).E[x_i].n_i$ by one.

Resort *decremently the experience list Q* in *Sol(s).E*

Until *Stop_Criterion ()* becomes *true*.

Fig 3. Pseudo-code for the Proposed Algorithm *CB-BHASQL*

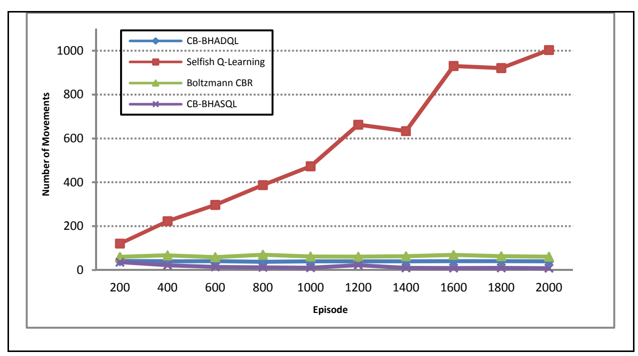


Fig4. Comparison of different methods in terms of the number of movements Needed for reaching to the optimal path in Grid World.

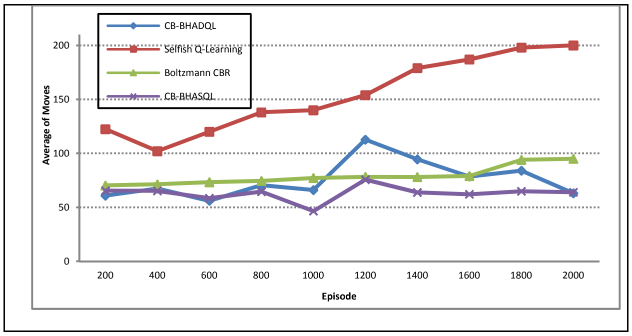


Fig5. Comparison of four Algorithms in terms of average of the number of moves to reaching optimal path in 200 runs in Grid World.

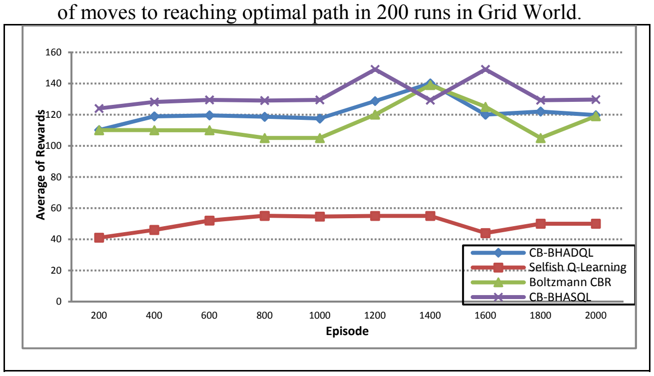


Figure 6. Comparison of Three Algorithms in term of the average of rewards gained in 200 runs in Grid World.

4.2. Experiments in Tunnel to Goal Games

In this section, we show results of our experiments in Tunnel to Goal Games.

Experiment 3. In this experiment, we compare the proposed algorithm (CB-BHASQL) in Tunnel to Goal Games environment (Fig 2.b) with the other algorithms in terms of the number of movements made by agent 1 to reach the optimal path in 2000 episode. Figure 7 illustrates the results of this experiment. Figure 8 shows the average results after 200 runs. From the result, it is evident that the CB-BHASQL algorithm has lower numbers of moves in comparison with the other algorithms.

Experiment 4. In this experiment, we compare the proposed algorithm (CB-BHASQL) in Tunnel to Goal Games environment (Fig 2.b) with the other algorithms in terms of the averaged reward received by agent 1during an episode. Figure 9 shows the result of this experiment. As it is seen (CB-BHASQL) algorithm outperforms the other algorithms in terms the average reward received during an episode.

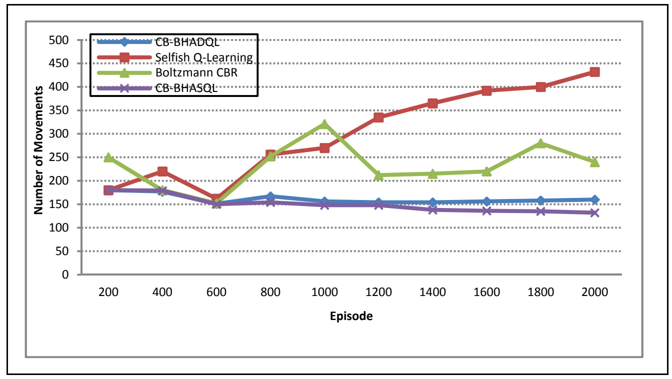


Fig7. Comparison of different methods in terms of the number of movements Needed for reaching to the optimal path in Tunnel to Goal.

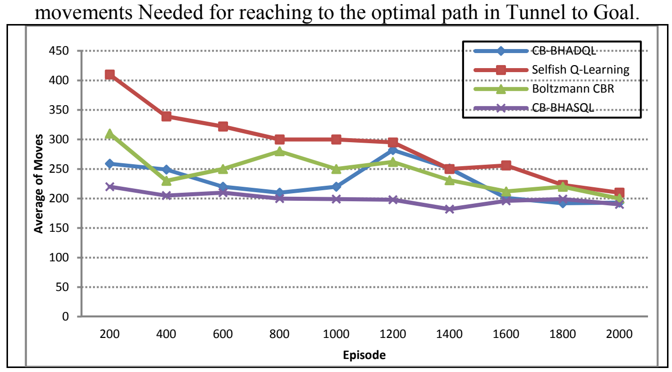


Fig8. Comparison of four Algorithms in terms of average of the number of moves to reaching optimal path in 200 runs in Tunnel to Goal.

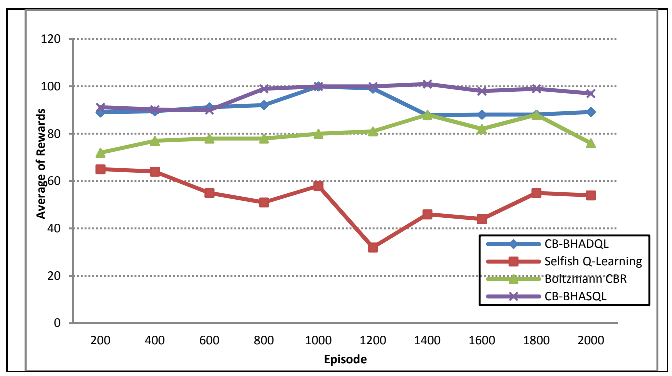


Fig 9. Comparison of Three Algorithms in term of the average of rewards gained in 200 runs in Tunnel to Goal.

5. Evaluation of the Algorithm's Behavior

5.1. Examination of the Behavior of the Proposed Algorithms

In this section, an analysis of the performance the proposed algorithm is conducted in which the advantage of the function π_2 *(S)* (EQ 7) is compared with π_1 *(S)* (EQ 4). We want to show that in the proposed method, π_2 *(S)* in comparison with π_1 *(S)* converges to the optimum solution with a higher rate. In other words, the rate of variation for

π_2 *(S)* in relation to *Q*, is more than the rate of variation for π_1 *(S)* in relation to *Q*.
To show the advantage of the behavior of the proposed action selection function, the *CB-BHASQL* algorithm was evaluated for state S_0 and action a_1 regarding to different values of *n* and the results below were gained:

Experiment 5. In this experiment, variations of π_2 (S) were evaluated in comparison with π_1 *(S)*. Figures 10-12 show these variations As it is seen, we note that regarding to the increase in *n*, the growth of π_2 *(S)* is much more than π_1 *(S)*.

Experiment 6. In this experiment, we study evaluation of variations for function *Q (S, a)* (EQ(3)) regarding to the increase in *n*. The results of this analysis are shown in Figures 11-13. As it is seen, we conclude that with increasing *n*, the value of function *Q (S, a)* also increases in Equation 3.

Experiment 7. In this experiment, we study evaluation of variations for π_1 *(S)* and π_2 *(S)* based on values for *Q (S, a)*. The results of this evaluation are shown in Figure 14. Looking at the diagram we note that with increasing value of *Q (S, a)*, the value of π_1 *(S)* increases. Since always $lim_{t\to\infty}$ *not (S, a)* = ∞, according to the result of Experiment 6, value of Q_t *(S, a)* increases and according to the result of Experiment 6, with increasing *n*, the function π_2 *(S)* grows faster than π_1 *(S)*. Based on the previous subjects, it is concluded that with increasing value of Q_t *(S, a)*, the function π_2 *(S)* must grow faster than π_1 *(S)*. Figure 15 shows the results.

5.2. Mathematical Analysis of the Functions Behavior

To facilitate the calculations, we rewrite functions π_1 *(S)* and π_2 *(S)* as EQ (9) and EQ (10) respectively.

$$\pi_1(S) = \frac{e^{\frac{Q}{\tau}}}{\sum_{j=1}^{m} e^{\frac{Q_j}{\tau}}} \tag{9}$$

$$\pi_2(S) = e^{nQ} \tag{10}$$

The variable rate of π_1 *(S)* in relation to *Q* with parameter $t = 0.05$, is shown in EQ (11).

$$\frac{\Delta\pi_1(S)}{\Delta Q} = \frac{1}{\tau}e^{\frac{1}{\tau}} = \frac{1}{0.05}e^{\frac{1}{0.05}} = 20e^{20Q} \tag{11}$$

The variable rate of π_1 *(S)* in relation to *Q* is shown in EQ (12).

$$\frac{\Delta\pi_2(S)}{\Delta Q} = \frac{d\pi_2(S)}{dn} \times \frac{dn}{dQ} \tag{12}$$
$$= Qe^{nQ} \times \frac{dn}{dQ}$$

Through the comparison of the EQ(11) and EQ(12), we conclude that the growth of the rate $\frac{\Delta\pi 1(S)}{\Delta Q}$ Is less than $\frac{\Delta\pi 2\,(S)}{\Delta Q}$.
In Equation 11, because *Q* is positive, the value of function $\frac{\Delta\pi 1\,(S)}{\Delta Q}$ is always positive. According to equation 3 and the diagram in Figure 11, with increasing *n*, the value of *Q (S, a)* always increases. Thus, $\frac{dn}{dQ} > 0$ and from the other hand, *n> 0* and *Q> 0*. So, $\frac{\Delta\pi 2\,(S)}{\Delta Q}$ is always positive.
In Equation 10, e^{20Q} is multiplied by the constant value 20. This is while in Equation 12, e^{nQ} is multiplied by variant *Q*. With increasing *n*, the value of *Q* increases. Thus, the growth rate of $\frac{\Delta\pi 1\,(S)}{\Delta Q}$ is less than $\frac{\Delta\pi 2(S)}{\Delta Q}$. The diagram in Figure 15 also supports the result gained.

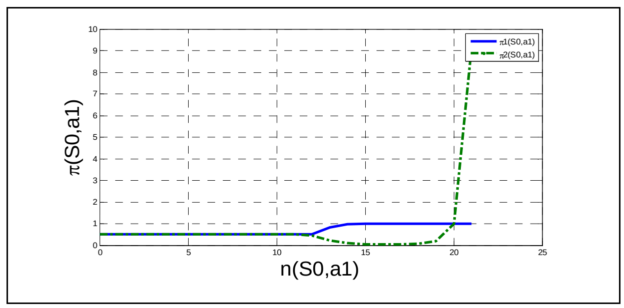


Fig 10. Examining the growth of π_2 *(S)* and π_1 *(S)* based on the different *n* values

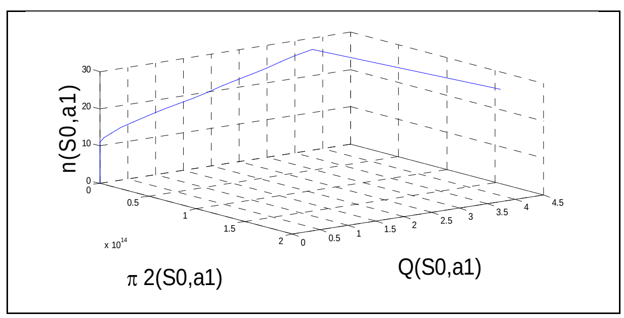


Fig 11. Examination of π_2*(S)* variations based on the different *n* values.

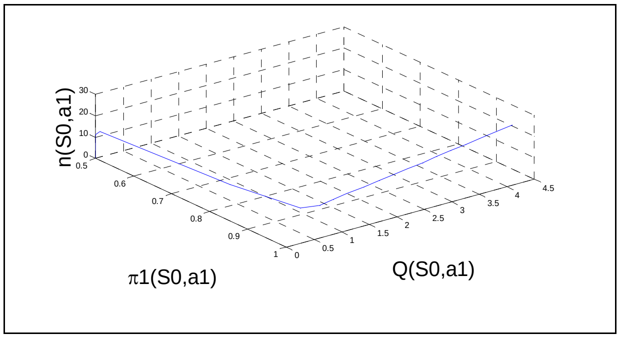


Fig 12. Examination of π_1 *(S)* Variations based on the Different *n* values.

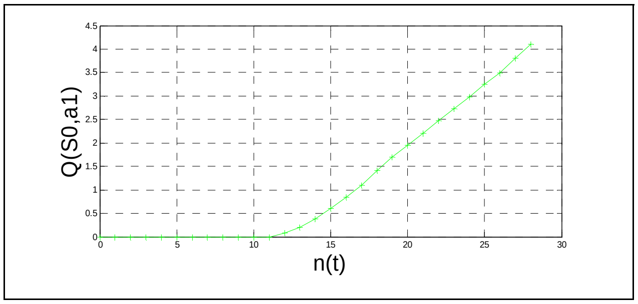


Fig 13. Examination of *Q (S, a)* Variations based on the Different *n* values.

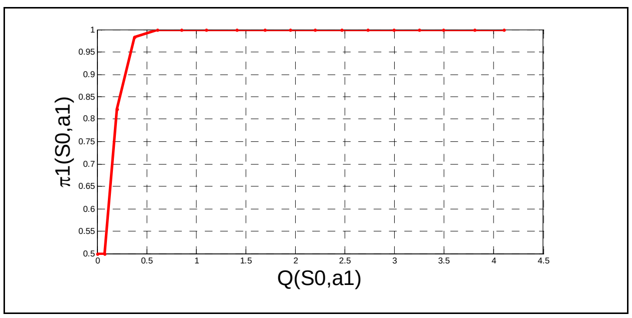


Fig 14. Examination of π_1 *(S)* Growth based on the Different *Q (S, a)* values.

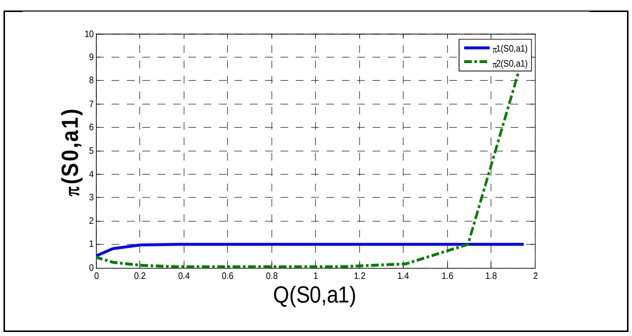


Fig 15. Examination of π_1 *(S)* and π_2 *(S)* Growth based on the Different *Q (S, a)* values

6. Conclusion

In this paper, a new hybrid model called *CB-BHAQL* has been introduced to solve Markov's games based on reinforcement learning and case base systems, in which a new function has been applied to select the action in each state. The results gained were compared with the current algorithms. Based on the results gained in two different environments, in comparison with *Selfish Q-Learning* and *Boltzmann's CBR* algorithms and *CB-BHADQL*, our proposed algorithm *CB-BHASQL* algorithm has a very high efficiency from the perspective of convergence rate to the optimum answer, average of total reward gained and the number of the movements needed for convergence to the optimal policy.

References

[1] R. A. C. Branchi, R. Raquel, R. L. D. Mantaras, " Imroving Reinforcement Learning by using Case Based Heuristics", Proceeding of the Int. Conference on Case Based Learning 2009 (ICCBR 2009), Springer , 2009.

[2] N. Vlassis, "A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence", 2007, Morgan and Claypool Publishers.

[3] C. Boutilier, "Sequential optimality and coordination in multi-agent systems", in: Proceedings of the 16th International joint conference on Artificial intelligence, 1999 , Vol. 1, Morgan Kaufmann Publishers Inc., Stockholm, Sweden.

[4] L. Bosniu, R. Babuska, and B. Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning", IEEE Transaction on System, Man, Cybern, 2008 ,vol. 38, pp. 156-171.

[5] B. Masoumi, M. R. Meybodi, "Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy", Journal of Expert Systems with Applications, July 2011, Vol 38, Issue 7, PP. 8105-8118.

[6] M. Lauer and M. Riedmiller, "An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems", in The 17th International Conference on Machine Learning San Francisco, CA, USA, 2000: Morgan Kaufmann Publishers Inc, pp. 535 – 542.

[7] A. G. Barto and R. S. Sutton, "*Reinforcement Learning: an introduction*", MIT Press, Cambridge, MA, 1998**.**

[8] X. Wang and T. Sandholm, "Reinforcement Learning to Play an Optimal Nash Equilibrium in Team Markov Games", in Advances in Neural Information Processing Systems, 2002, vol. 15: MIT Press, pp. 1571-1578, 2002,

[9] F. S. Melo, M. I. Ribeiro, "Reinforcement Learning with Function Approximation for Cooperative Navigation Tasks", IEEE International Conference on Robotics and A Utomation Pasadena, CA, USA, May 2008, pp. 3321-2237.

[10] M. Lauer and M. Riedmiller, "Reinforcement Learning for Stochastic cooperative Multi-agent Systems", In Proceeding of AAMAS 2004, New York, NY, ACM Press, pp. 1514-1515.

[11] R. A. C. Bianchi, C. H. C. Ribeiro, A. H. R. Costa, " *Accelerating autonomous learning by using a heuristic selection of actions*", Journal of Heuristis , , 2008, Vol. 2, pp.135-168.

[12] R. A. C. Bianchi, C. H. C. Ribeiro, A. H. R. Costa, "Heuristic selection of actions in multi agent reinforcement

learning", 20th International conference on Artificial Intelligence, India , Jan 2007, pp.690-695.

[13] L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley and Sons, New York, 1994.

[14] R. S. Sutton, A. G. Barto, "Reinforcement Learning : An Introduction", MIT Press, 1998.

[15] J. F. Nash, "Non-cooperative Games", Annals of Mathematics, , 1951, Vol. 54, pp. 286–295.

[16] A. M. Fink, Equilibrium in a Stochastic N-person Game, Journal of Science in Hiroshima University, Series A-I, 1964, Vol. 28, pp. 89–93.

[17] A. Aamodt; E. Plaza, "Case-Based Reasoning: Foundational Issues", Methodological Variations and System Approaches AI Communications, IOS Press, 1994, Vol. 7, No. 1, pp. 39-59.

[18] R. Bergman; "Engineering Applications of Case Based Reasoning", Journal of Engineering Applications of Artificial Intelligence, 1999 , Vol. 12, pp.805.

[19] Gabel, T. And Riedmiller, M., "CBR for state value function Approximation in Reinforcement Learning", Proceeding of the Inter. Conference on Case Based Learning 2005 (ICCBR 2005) , Springer , Chicago, USA.

Sara Esfandiari received her BS degree in Computer Engineering from the azad University, Tehran, Iran., in 2006 and MS degree in Computer Engineering in 2011 from the azad University, Qazvin, Iran. Her research interests include Learning systems, multi agent systems, multi agent learning, Data Mining, parallel algorithms.

Behrooz Masoumi received his BS and MS degrees in Computer Engineering in 1995 and 1998, respectively. He also received his PhD degrees in Computer Engineering from the Science and Research University, Tehran, Iran., in 2011. He joined the faculty of Computer and IT Engineering Department at Qazvin Azad University, Qazvin, Iran, in 1998. His research interests include learning systems, multi-agent systems, multi-agent learning, and soft computing.

Mohammadreza Meybodi received his BS and MS degrees in Economics from the Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received his MS and PhD degrees in Computer Science from the Oklahoma University, U.S.A., in 1980 and 1983, respectively. Currently he is a Full Professor in the Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to his current position, he worked from 1983 to 1985 as an Assistant Professor at Western Michigan University, and from 1985 to 1991 as an Associate Professor at Ohio University, U.S.A. His research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing, and software development.

Abdolkarim Niazi is currently PhD Student in Mechanical Engineering-Manufacturing engineering at Technical University of Malaysia from 2010 up to now. His research interests include Condition monitoring, Tools Condition monitoring, Tool wear and tool vibration, Advance and Automated Manufacturing Systems, Artificial Neural Networks.